

Математика в Python

2018

План

Введение

DataFrame

Линейная алгебра

Полиномы

Дифференциальные уравнения

Численные методы

- Минимизация функции

- Метод наименьших квадратов

- Интерполяция

- Интегрирование

- Дифференцирование

Теория вероятностей и статистика

- Гистограмма

- Корреляция и диаграмма рассеивания

- Диаграмма размаха

Символьные вычисления

О быстройдействии

Ссылки и литература

Outline

Введение

DataFrame

Линейная алгебра

Полиномы

Дифференциальные уравнения

Численные методы

- Минимизация функции

- Метод наименьших квадратов

- Интерполяция

- Интегрирование

- Дифференцирование

Теория вероятностей и статистика

- Гистограмма

- Корреляция и диаграмма рассеивания

- Диаграмма размаха

Символьные вычисления

О быстройдействии

Ссылки и литература

Среда разработки Jupyter



jupyter Untitled Last Checkpoint: 6 minutes ago (unsaved changes)



File Edit View Insert Cell Kernel Help

Trusted

Python

+

```
In [7]: from math import *  
import numpy as np  
from matplotlib.pyplot import *
```

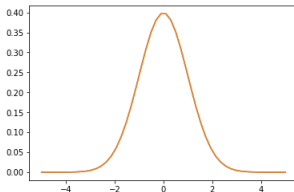
```
In [6]: math.sin( math.pi )
```

```
Out[6]: 1.2246467991473532e-16
```

```
In [14]: def norm_pdf(x):  
return 1/sqrt(2*pi)*exp(-x**2/2)
```

```
X = np.linspace(-5, 5)  
Y = [ norm_pdf(x) for x in X ]
```

```
plot(X,Y)  
show()
```



Среда разработки Jupyter

- ▶ Jupyter позволяет хранить вместе код, изображения, комментарии, формулы и графики
- ▶ Jupyter поддерживает множество языков программирования
- ▶ может быть легко запущен на любом сервере, необходим только доступ по ssh или http.
- ▶ Поддержка LaTeX
- ▶ Блочный запуск

Jupyter входит в дистрибутив **Anaconda**.

Помимо Jupyter туда входят популярные математические пакеты для Python, не включенные в стандартную библиотеку языка.

Доклад: как использовать Jupyter на 100%

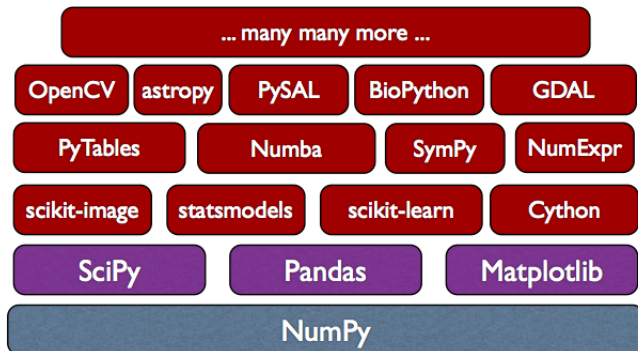
<https://www.youtube.com/watch?v=q4d-hKCpTEc>

Некоторые популярные математические пакеты

- ▶ **numpy** - работа с матрицами и многомерными массивами; оптимизирован для работы с многомерными массивами; Написан на C и Python.
- ▶ **scipy** - научные и инженерные вычисления, использует numpy;
- ▶ **sympy** - символьные вычисления;
- ▶ **matplotlib** - построение графиков и диаграмм;
- ▶ **seaborn** - визуализация статистических данных, эстетичнее чем matplotlib;
- ▶ **mpld3** - использование D3.js для построения интерактивных matplotlib графиков в окне браузера;
- ▶ **pandas** - анализ данных: статистики, регрессия, визуализация и т.п.

Популярные математические пакеты

Многие пакеты построены на основе NumPy



Outline

Введение

DataFrame

Линейная алгебра

Полиномы

Дифференциальные уравнения

Численные методы

- Минимизация функции

- Метод наименьших квадратов

- Интерполяция

- Интегрирование

- Дифференцирование

Теория вероятностей и статистика

- Гистограмма

- Корреляция и диаграмма рассеивания

- Диаграмма размаха

Символьные вычисления

О быстродействии

Ссылки и литература

DataFrame

Наборы значений часто удобнее хранить не в списках или динамических массивах `numpy ndarray`, а в виде таблиц **DataFrame** (из пакета *pandas*).

Этот тип данных помимо хранения значений даёт возможность делать простые запросы, более гибко при обращении с данными. Например можно выбирать строки, поля которых содержат только положительные значения.

DataFrame

Создание DataFrame из словаря¹

```
import pandas as pd
```

```
# создание пустой таблицы
```

```
D0 = pd.DataFrame()
```

```
# создание пустой таблицы с названиями столбцов
```

```
D1 = pd.DataFrame( columns = ['x', 't'] )
```

```
# создание таблицы с данными из словаря
```

```
D = pd.DataFrame( { 'x': [1,2,3],  
                    't': [0, 0.1, 0.2] } )
```

x, t - заголовки столбцов таблицы

¹словарь в Python задаётся в фигурных скобках; для создания корректного DataFrame ключи словаря должны иметь строковый тип»

DataFrame

Запросы с условием:

```
# выбрать только те строки таблицы, где поле x >= 2  
D[ D['x'] >= 2 ]
```

```
# выбрать только те строки таблицы,  
# где поле x >= 2 и t != 0  
D[ (D['x'] >= 2) & (D['t'] != 0) ]
```

DataFrame

Запросы особенно полезны если в таблице есть категориальные данные. Обычно они представлены строками, например название групп A, B,C; городов Moscow, London; отдельных бинарных признаков да/нет и т.п.

DataFrame

Добавление новых столбцов в DataFrame похоже на добавление новой записи в словарь

добавление столбца y с набором значений

```
D['y'] = [1,4,9]
```

Аналогичный синтаксис используется для доступа к столбцам и ячейкам

```
D['y']    # -> 1,4,9
```

```
D['y'][1]  # -> 4
```

DataFrame

Обращение по номеру

2-я строка (счёт начинается с 0)

```
D.iloc[2] # t = 0.2, x = 3.0
```

2-я строка, нулевой столбец

```
D.iloc[2, 0] # t = 0.2
```

DataFrame

Сохранение в CSV файл

```
DataFrame.to_csv(filename= ... , sep=', ')
```

filename - имя файла

sep - разделитель для чисел в строке

DataFrame

Загрузка из CSV файла

```
pandas.read_csv(filename = ... , sep=',') -> DataFrame
```

filename - имя файла

sep - разделитель для чисел в строке

DataFrame

Статистическая информация о данных

D.describe()

	Vx	Vy	x	y
count	3.0	3.000000	3.0	3.00
mean	1.0	0.733333	2.0	0.20
std	0.0	0.251661	1.0	0.10
min	1.0	0.500000	1.0	0.10
25%	1.0	0.600000	1.5	0.15
50%	1.0	0.700000	2.0	0.20
75%	1.0	0.850000	2.5	0.25
max	1.0	1.000000	3.0	0.30

DataFrame

Пакеты для визуализации данных (например seaborn) могут использовать DataFrame непосредственно в качестве источника данных, а не отдельные списки значений.

Outline

Введение

DataFrame

Линейная алгебра

Полиномы

Дифференциальные уравнения

Численные методы

- Минимизация функции

- Метод наименьших квадратов

- Интерполяция

- Интегрирование

- Дифференцирование

Теория вероятностей и статистика

- Гистограмма

- Корреляция и диаграмма рассеивания

- Диаграмма размаха

Символьные вычисления

О быстройдействии

Ссылки и литература

Линейная алгебра

```
import numpy as np

# Создание матрицы
A = np.array([ [1,2,3],
               [3,2,1],
               [2,1,3] ] )
```

Матрица A - это двумерный массив (список), только в обёртке numpy - ndarray.

Линейная алгебра

умножение на число

A * 3.14

```
> array([[ 3.14,  6.28,  9.42],  
        [ 9.42,  6.28,  3.14],  
        [ 6.28,  3.14,  9.42]])
```

Умножение на вектор

A * [2,3,4]

```
array([[ 2,  6, 12],  
       [ 6,  6,  4],  
       [ 4,  3, 12]])
```

Причём здесь вектор не обязательно должен быть типом numpy.

Линейная алгебра

Сложение матриц

$A + B$

Умножение матриц

$A @ B$

или

`np.dot(A,B)`

Обратная матрица

`np.linalg.inv(A)`

Линейная алгебра

Решение СЛАУ

Настройка вывода. Число знаков после запятой - 4.

не выводить числа в экспоненциальной форме

```
numpy.set_printoptions(precision=4, suppress=True)
```

```
A = np.matrix([
```

```
    [1, 2, 3],
```

```
    [3, 2, 1],
```

```
    [2, 3, 1]])
```

```
B = np.matrix([[1], [2], [3]])
```

```
X = np.linalg.solve(A, B)
```

```
matrix([[ 0.0833],
```

```
        [ 1.0833],
```

```
        [-0.4167]])
```

Стоит обратить внимание на то, что вектор-столбец определяется как матрица из одного столбца, а не как список.

DataFrame vs numpy.ndarray

- ▶ производительность чаще всего ndarray выше чем у DataFrame
- ▶ для ndarray реализованы операторы матричной алгебры и векторные варианты математических функций (например \sin , \ln и т.д.)
- ▶ индексация и обращение к элементам DataFrame более гибкие чем у ndarray
- ▶ но доступ к элементам занимает больше времени

Outline

Введение

DataFrame

Линейная алгебра

Полиномы

Дифференциальные уравнения

Численные методы

- Минимизация функции

- Метод наименьших квадратов

- Интерполяция

- Интегрирование

- Дифференцирование

Теория вероятностей и статистика

- Гистограмма

- Корреляция и диаграмма рассеивания

- Диаграмма размаха

Символьные вычисления

О быстройдействии

Ссылки и литература

Полиномы

```
import numpy as np
# полином одной переменной
p1 = np.poly1d([2.1, -0.3, -2])

print(p1)
# 2.1 x^2 - 0.3 x + 7

# вычисление значения полинома
p1(10) # 214.0

# корни полинома (p1 = 0)
p1.r
# [ 1.04993917 -0.90708203 ]
```

Полиномы

```
from matplotlib.pyplot import *
```

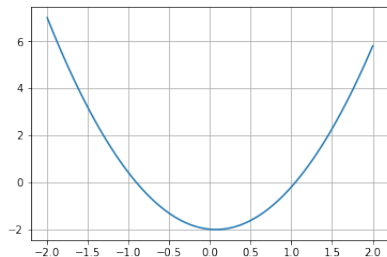
```
x = np.linspace(-2, 2, 100)
```

```
y = [p1(x) for x in x]
```

```
plot(x,y)
```

```
grid(True)
```

```
show()
```



Outline

Введение

DataFrame

Линейная алгебра

Полиномы

Дифференциальные уравнения

Численные методы

- Минимизация функции

- Метод наименьших квадратов

- Интерполяция

- Интегрирование

- Дифференцирование

Теория вероятностей и статистика

- Гистограмма

- Корреляция и диаграмма рассеивания

- Диаграмма размаха

Символьные вычисления

О быстройдействии

Ссылки и литература

Построение множества решений ДУ

Пример

Общим решением дифференциального уравнения

$$\dot{x} = x$$

где $\dot{x} \equiv \frac{dx}{dt}$,

является функция

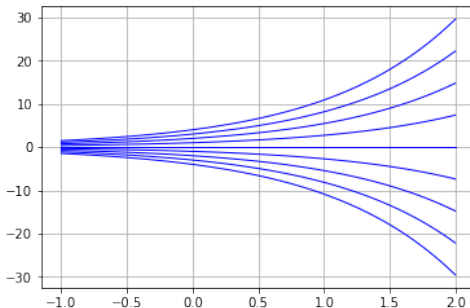
$$x(t) = c_1 e^t$$

где C - константа интегрирования. $x(t)$ на самом деле представляет собой семейство интегральных кривых.

Бывает полезно изобразить множество решений ДУ на плоскости для различных значений постоянной

Построение множества решений ДУ

Пример



набор значений t (ось абсцисс) для построения графика

```
t = np.linspace(-1,2)
```

решение ДУ. Принимает t , возвращает набор соответствующих ордина

```
x = lambda C, t: C*np.exp(t)
```

```
grid()
```

перебор значений константы интегрирования

```
for C in np.linspace(-4, 4, 9):
```

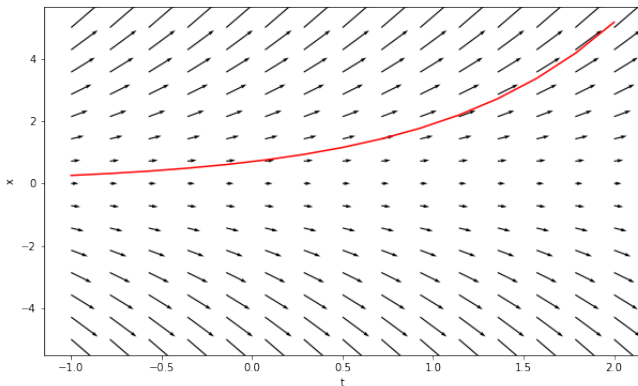
```
    plot(t, x(C,t),color='blue', lw = 1)
```

Поле направлений

Пример

Другой вариант графически представить множество решений дифференциального уравнения - построить поле направлений.

Для построения векторных полей используется функция `quiver`



Красная кривая - частное решение ДУ

Поле направлений

Поле направлений (векторное поле) можно построить не решая дифференциальное уравнение

При этом векторное поле даёт представление о том как будут выглядеть интегральные кривые - решения дифференциального уравнения.

Построение множества решений ДУ

Пример

```
def dxdt(x,t):  # правая часть ДУ  $x' = f(x,t)$ 
    return x

def x_func(C,t):  # решение ДУ
    return C*np.exp(t)

n = 15
# для построения поля направлений понадобится набор точек
# в которых и будут изображены вектора
xpoints, ypoints = np.linspace(-1,2, n), np.linspace(-5, 5, n)
xgrid,  ygrid = np.meshgrid(xpoints, ypoints)

dt = np.ones(xgrid.shape)  # первая компонента векторов
dx = dxdt(ygrid, 0)         # вторая компонента векторов
# в функцию определяющую правую часть уравнения
# передаются на место t - абсциссы, на место x - ординаты

plt.quiver(xgrid,ygrid, dt,dx, width = 0.002, angles='xy')
plot(xpoints, x_func(0.7, xpoints), linewidth = 1.5, color='red')
```

Численное решение однородных ДУ

Для численного решения ОДЕ требуется перевести его к виду

$$y' = f(y, t)$$

y и f могут быть векторами.

Для решения используется функция `odeint`:

```
from scipy.integrate import odeint
```

`odeint($f(y, t)$, y_0 , набор t)` → набор y

Функция возвращает набор значения искомой функции y для заданного набора значений её аргумента t и начального значения y_0

Документация:

docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.odeint.html

readme проекта `scipy` с указанием используемых методов решения ДУ

github.com/scipy/scipy/blob/v0.19.0/scipy/integrate/odepack/readme

Численное решение однородных ДУ

`odeint($f(y, t)$, y_0 , набор t) \rightarrow набор y`

Функция $f(y, t)$ должна иметь в точности такой же порядок параметров, но не обязательно чтобы все они использовались

```
def f(y, t):  
    return ...
```

Численное решение однородных ДУ

```
import numpy as np
from scipy.integrate import odeint

# правая часть ДУ вида:  $y' = f(y, t)$ 
def dydt(y, t):
    return y

# набор значений, для которого будет вычислено значение функции y
t = np.linspace( 0, 2, 30)
# начальное значение
y0 = 1
y = odeint (dydt, y0, t) # получим значения y
# полученный массив - это матрица-столбец
# поэтому преобразуем в массив
y = y.flatten()
print(y)
plt.plot( t, y, '-.')
plt.show()
```

Численное решение однородных ДУ

уравнения второго порядка

Пример: уравнение колебания маятника с учётом трения

$$\theta''(t) + b * \theta'(t) + c * \sin(\theta(t)) = 0$$

θ - угол отклонения маятника от положения равновесия

- ▶ Для решения ДУ второго порядка его нужно представить системой из двух ДУ первого порядка.
- ▶ Сделаем замену понижающую порядок уравнения $u = \theta'$, таким образом добавив ещё одно.
- ▶ Приведём получившиеся уравнения к виду $y' = f(y, t)$

$$\begin{cases} \theta' = u \\ u' = -bu - c \sin(\theta) \end{cases}$$

Численное решение однородных ДУ

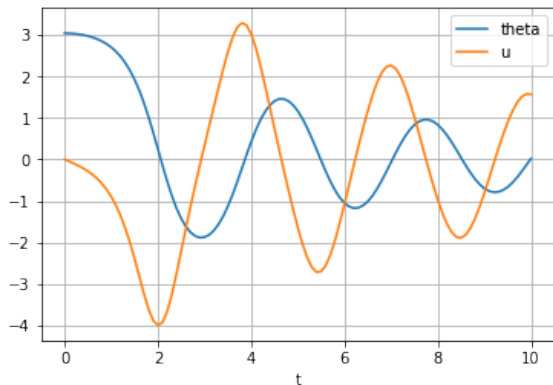
уравнения второго порядка

$$\begin{cases} \theta' = u \\ u' = -bu - c \sin(\theta) \end{cases}$$

```
def de(y, t, b, c):  
    th, u = y  
    # список из правых частей уравнений  
    return [u, -b * u - c*sin(th)]  
  
b = 0.25  
c = 5.0  
# уравнений два, поэтому и начальных значений тоже  
y0 = [np.pi - 0.1, 0]  
t = np.linspace(0,10, 101)  
yy = odeint(de, y0, t, args=(b,c))  
plt.plot(t, yy[:,0], label='theta')  
plt.plot(t, yy[:,1], label='u')  
xlabel('t')  
plt.legend(loc = 0)  
plt.grid(True)  
plt.show()
```

Численное решение однородных ДУ

уравнения второго порядка



Численное решение однородных ДУ

Фазовый портрет

Используя то же решение построим фазовый портрет системы уравнений. Вдоль осей координат будем отчитывать θ и u , t в этом случае будет лишь параметром.

```
def de(y, t, b, c):  
    th, u = y  
    # список из правых частей уравнений ()  
    return [u, -b * u - c*np.sin(th)]  
  
b = 0.25  
c = 5.0  
# уравнений два, поэтому и начальных значений тоже  
y0 = [np.pi - 0.1, 0]  
t = np.linspace(0,10, 101)  
yy = odeint(de, y0, t, args=(b,c))  
plt.plot(yy[:,0], yy[:,1]) # кривая  
plt.plot(yy[0,0], yy[0,1], 'o', label = 'start') # начало  
xlabel(r'$\theta$')  
ylabel('u')  
plt.grid(True)  
plt.show()
```

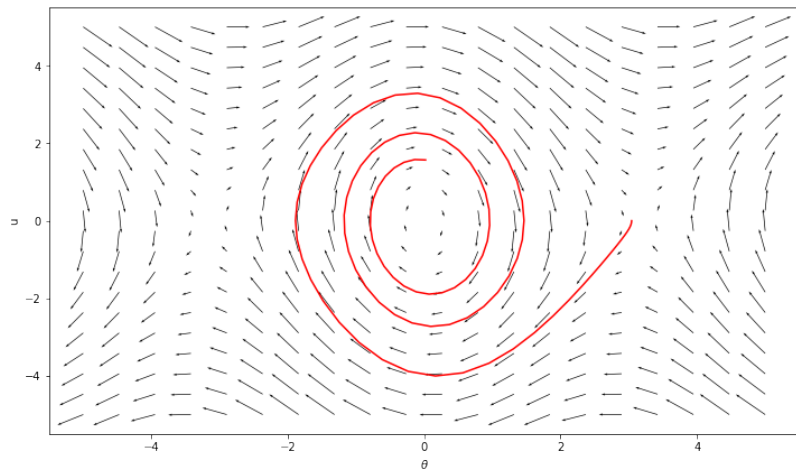
Численное решение однородных ДУ

Интегральные кривые и поле направлений

```
def de(y, t, b, c):  
    th, u = y  
    return [u, -b * u - c*np.sin(th)]  
  
n = 20  
xpoints, ypoints = np.linspace(-5,5, n), np.linspace(-5,5, n)  
xgrid,ygrid = np.meshgrid(xpoints, ypoints)  
t = np.linspace(0,10, 100)  
# начальные значения: угол отклонения - чуть меньше 180 градусов  
y0 = np.pi-0.1, 0 # начальная скорость 0  
yy = odeint(de, y0, t, args=(0.25,5))  
  
# вектора  
dth,dv = de([xgrid,ygrid], t, b=0.25, c=5)  
plt.xlabel(r'$\theta$')  
plt.ylabel('u')  
plt.quiver(xgrid, ygrid, dth,dv,  
            headwidth = 3, width = 0.001, angles='xy')  
plt.plot(yy[:,0], yy[:,1], linewidth = 1.5, color='red')
```

Численное решение однородных ДУ

Интегральные кривые и поле направлений



Outline

Введение

DataFrame

Линейная алгебра

Полиномы

Дифференциальные уравнения

Численные методы

- Минимизация функции

- Метод наименьших квадратов

- Интерполяция

- Интегрирование

- Дифференцирование

Теория вероятностей и статистика

- Гистограмма

- Корреляция и диаграмма рассеивания

- Диаграмма размаха

Символьные вычисления

О быстродействии

Ссылки и литература

Outline

Введение

DataFrame

Линейная алгебра

Полиномы

Дифференциальные уравнения

Численные методы

- Минимизация функции

- Метод наименьших квадратов

- Интерполяция

- Интегрирование

- Дифференцирование

Теория вероятностей и статистика

- Гистограмма

- Корреляция и диаграмма рассеивания

- Диаграмма размаха

Символьные вычисления

О быстройдействии

Ссылки и литература

Минимизация функции

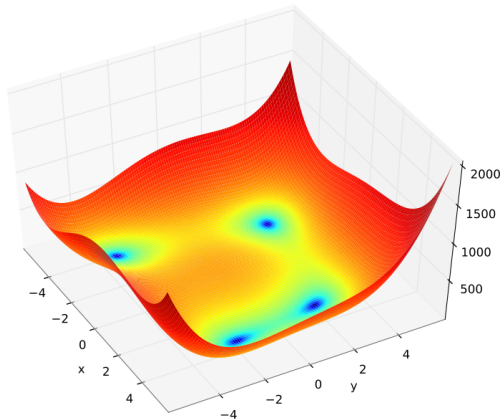
```
from scipy.optimize import minimize
x0 = 100 # начальное значение
minimize(lambda x: (x-3)*(x-3) - 5, x0)
```

Вывод:

```
nfev: 12
jac: array([ 0.])
message: 'Optimization terminated successfully.'
fun: -5.0
success: True
x: array([ 3.])
njev: 4
hess_inv: array([[ 0.5]])
status: 0
```

Минимум функции: $f(3) = -5$

Тестовые функции для оптимизации



см. также список тестовых функций для оптимизации

Outline

Введение

DataFrame

Линейная алгебра

Полиномы

Дифференциальные уравнения

Численные методы

Минимизация функции

Метод наименьших квадратов

Интерполяция

Интегрирование

Дифференцирование

Теория вероятностей и статистика

Гистограмма

Корреляция и диаграмма рассеивания

Диаграмма размаха

Символьные вычисления

О быстройдействии

Ссылки и литература

Метод наименьших квадратов

Нелинейный метод наименьших квадратов для поиска параметров известной функции, если её значения заданы таблично (экспериментальные данные)

<https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve>

Outline

Введение

DataFrame

Линейная алгебра

Полиномы

Дифференциальные уравнения

Численные методы

Минимизация функции

Метод наименьших квадратов

Интерполяция

Интегрирование

Дифференцирование

Теория вероятностей и статистика

Гистограмма

Корреляция и диаграмма рассеивания

Диаграмма размаха

Символьные вычисления

О быстройдействии

Ссылки и литература

Численные методы

Интерполяция

Интерполяция - нахождение промежуточных значений величины по имеющемуся дискретному набору известных значений.

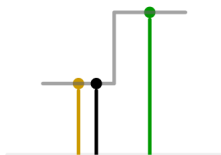
Слайн (spline) — функция, область определения которой разбита на конечное число отрезков, на каждом из которых она совпадает с некоторым алгебраическим многочленом (полиномом).

Максимальная из степеней использованных полиномов называется **степенью сплайна**.

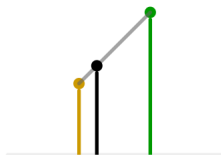
сплайн — это кусочно заданная функция

Слайн

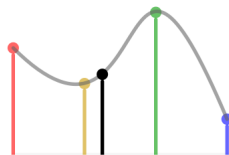
Примеры интерполяции для отдельных участков.



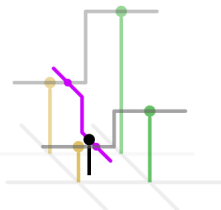
1D nearest-neighbour



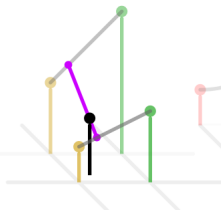
Linear



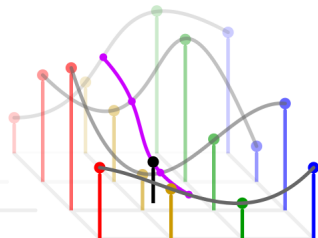
Cubic



2D nearest-neighbour



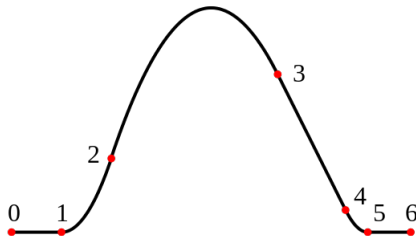
Bilinear



Bicubic

График кучочно заданной функции

Сплайн второй степени



- ▶ 0-1 Прямая
- ▶ 1-2 Парабола
- ▶ 2-3 Парабола
- ▶ 3-4 Прямая
- ▶ 4-5 Парабола
- ▶ 5-6 Прямая

Интерполяция

Интерполяция функции одной переменной

```
from scipy.interpolate import interp1d
```

Возможна интерполяция сплайном первой, второй и третьей степени²

- ▶ linear
- ▶ quadratic
- ▶ cubic

²см. другие способы в документации

Интерполяция

Пусть функция задана в табличном виде - набором значения X и Y .

Задача - определить значение функции для X , не перечисленного в таблице.

Применем для этого интерполяцию, построив функцию в аналитическом виде, которая будет проходить наиболее близко к заданным точкам.

Пример 1

```
from scipy.interpolate import interp1d
```

```
X = [1,2,3,4,5,6,7]
```

```
Y = [1,4,9,16,25,36,49]
```

```
func = interp1d(X, Y, kind='cubic')
```

func - функция определённая на отрезке от $\min(x)$ до $\max(x)$.

kind - тип интерполяции (см. справку)

В примере использована интерполяция сплайном третьего порядка.

Теперь можно вычислять значение функции в любой точке.

```
func(5.3)
```

```
# array(28.090000000000003)
```

Пример 2

```
# таблично заданная функция
X = [1,2, 3, 8, 9, 12]
Y = [1,2,2.5, 20, 42, 30]

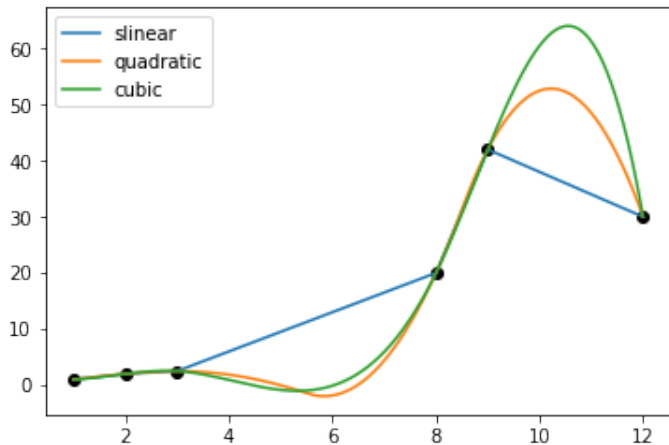
# создание сплайнов
f1 = interp1d(X,Y,'slinear')
f2 = interp1d(X,Y,'quadratic')
f3 = interp1d(X,Y,'cubic')

# набор точек для интерполяции
X0 = np.linspace(min(X), max(X), 1000)
Y1 = [f1(x) for x in X0]
Y2 = [f2(x) for x in X0]
Y3 = [f3(x) for x in X0]

plot(X,Y, 'o', color='black')
plot(X0, Y1, label='slinear')
plot(X0, Y2, label='quadratic')
plot(X0, Y3, label='cubic')

legend(loc='best')
show()
```

Пример 2



Пример 3

Интреполяция функции задающей поверхность

```
x = np.array([-5.0, -1.7, 1.7, 5.0])
y = np.array([-5.0, -1.7, 1.7, 5.0])
z = np.array([[ 50. , 27.8, 27.8, 50. ],
               [ 27.8, 5.6 , 5.6 , 27.8],
               [ 27.8, 5.6 , 5.6 , 27.8],
               [ 50. , 27.8, 27.8, 50. ]])

f = interp2d(x, y, z, kind='cubic')
```

Примеры использования модуля interpolate:
docs.scipy.org/doc/scipy-0.14.0/reference/tutorial/interpolate.html

Outline

Введение

DataFrame

Линейная алгебра

Полиномы

Дифференциальные уравнения

Численные методы

Минимизация функции

Метод наименьших квадратов

Интерполяция

Интегрирование

Дифференцирование

Теория вероятностей и статистика

Гистограмма

Корреляция и диаграмма рассеивания

Диаграмма размаха

Символьные вычисления

О быстройдействии

Ссылки и литература

Интегрирование

```
from scipy.integrate import quad
```

```
# интеграл от sin(x) в пределах от 0 до pi/2
```

```
quad(lambda x: sin(x): 0, pi/2)
```

```
>>> (0.9999999999999999, 1.1102230246251564e-14)
```

Outline

Введение

DataFrame

Линейная алгебра

Полиномы

Дифференциальные уравнения

Численные методы

Минимизация функции

Метод наименьших квадратов

Интерполяция

Интегрирование

Дифференцирование

Теория вероятностей и статистика

Гистограмма

Корреляция и диаграмма рассеивания

Диаграмма размаха

Символьные вычисления

О быстройдействии

Ссылки и литература

Дифференцирование

```
from scipy.misc import derivative  
derivative(lambda t: -3/(t+2), 2, dx=1e-6)
```

```
0.18750000002620837
```

Outline

Введение

DataFrame

Линейная алгебра

Полиномы

Дифференциальные уравнения

Численные методы

Минимизация функции

Метод наименьших квадратов

Интерполяция

Интегрирование

Дифференцирование

Теория вероятностей и статистика

Гистограмма

Корреляция и диаграмма рассеивания

Диаграмма размаха

Символьные вычисления

О быстройдействии

Ссылки и литература

Некоторые распределения

Распределения случайных величин представлены соответствующими классами

```
from scipy.stats import uniform # равномерное распр.  
from scipy.stats import norm # нормальное распр.  
from scipy.stats import f # распределение Фишера  
from scipy.stats import t # распред. Стьюдента (t распред)  
from scipy.stats import chi2 # распред. Хи-квадрат  
from scipy.stats import poisson # распред. Пуассона
```

Классы представляющие распределения (см. предыдущий слайд) имеют общие методы. Например создание выборки или вычисление функции распределения.

Смысл параметров (loc, scale, ...) большинства этих методов зависит от вида распределения и описан в документации класса.

Параметр *loc* обычно отвечает за *положение* кривой распределения. Этот параметр обычно связан с математическим ожиданием. например для нормального распределения он должен быть равен математическому ожиданию - положению пика на кривой распределения

scale - параметр масштаба, определяет форму кривой. для нормального распределения параметр равен среднеквадратическому отклонению

Некоторые методы

- ▶ `cdf(self, x, *args, **kwargs)`
Cumulative distribution function - функция распределения ($F(x)$)
- ▶ `pdf(self, x, *args, **kwargs)`
Probability density function at x - функция плотности ($f(x)$)

Некоторые методы

- ▶ `rvs(self, *args, **kwargs)`
Random variates of given type. параметр size определяет размер (размерность) выборки.
- ▶ `isf(self, q, *args, **kwargs)`
Inverse survival function (inverse of 'sf') at q
Вычисления аргумента функции распределения $F(x)$ по её значению q.

Генерирование значений случайной величины

Метод `rvs` возвращает `numpy array`.

Сделать 10 значений (выборку из 10 значений)

```
norm.rvs( size = 10 )
```

Создать массив из 10 пар значений

```
norm.rvs( size = (10, 2) )
```


Генерирование значений случайной величины

Сделать выборку нормально распределённой величины с математическим ожиданием 172 и стандартным отклонением 6

```
x = norm.rvs(loc = 172, scale = 6, size = 100)
```

По выборке легко построить гистограмму:

```
seaborn.distplot(x)
```

Числовые характеристики случайной величины

```
T = expon.rvs(scale = 5, size = 1000)
```

```
describe(T)
```

```
DescribeResult(nobs=1000,  
minmax=(0.005752334954425684, 36.12550987774781),  
mean=4.861700730309254,      # среднее значение  
variance=24.71604656698484,  # дисперсия  
skewness=2.0848488433691847, # коэф. асимметрии  
kurtosis=5.97505931499302)  # коэф. эксцесса
```

docs.scipy.org/doc/scipy/reference/generated/scipy.stats.describe.html

Outline

Введение

DataFrame

Линейная алгебра

Полиномы

Дифференциальные уравнения

Численные методы

Минимизация функции

Метод наименьших квадратов

Интерполяция

Интегрирование

Дифференцирование

Теория вероятностей и статистика

Гистограмма

Корреляция и диаграмма рассеивания

Диаграмма размаха

Символьные вычисления

О быстройдействии

Ссылки и литература

Гистограмма

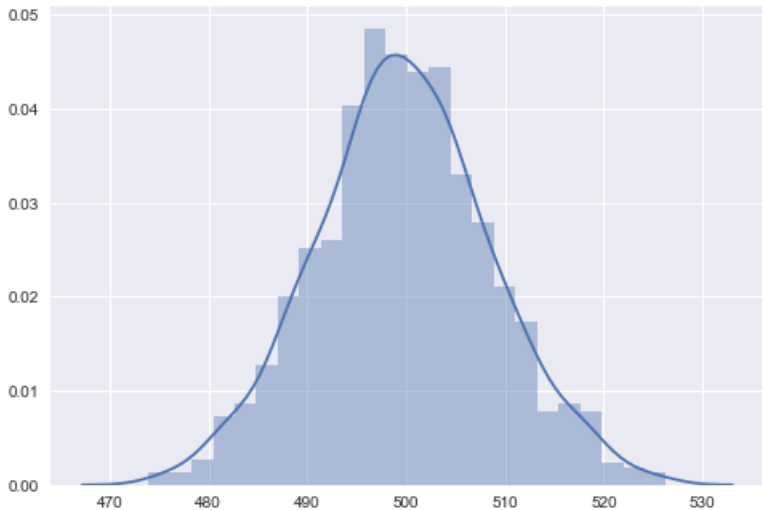
```
from random import random
import seaborn
from matplotlib.pyplot import show

X = [ sum( [random() for i in range(1000)] )
      for j in range(1000)]

# подготовим гистограмму и кривую распределения
seaborn.distplot(X)

show()
```

Гистограмма



Некоторые распределения

Вычислить вероятность того, что рост наугад выбранного человека будет меньше 180 см.

Средний рост людей 172 см.

Стандартное отклонение 7 см.

```
from scipy.stats import norm  
norm.cdf( (180 - 172) / 7 )  
0.87345104552644226
```

Сколько людей на Земле имеют рост меньше 180?

```
norm.cdf( (180 - 172) / 7 ) * 7.6e9  
6 638 227 946
```

Outline

Введение

DataFrame

Линейная алгебра

Полиномы

Дифференциальные уравнения

Численные методы

Минимизация функции

Метод наименьших квадратов

Интерполяция

Интегрирование

Дифференцирование

Теория вероятностей и статистика

Гистограмма

Корреляция и диаграмма рассеивания

Диаграмма размаха

Символьные вычисления

О быстройдействии

Ссылки и литература

Коэффициент корреляции

```
import seaborn # для визуализации

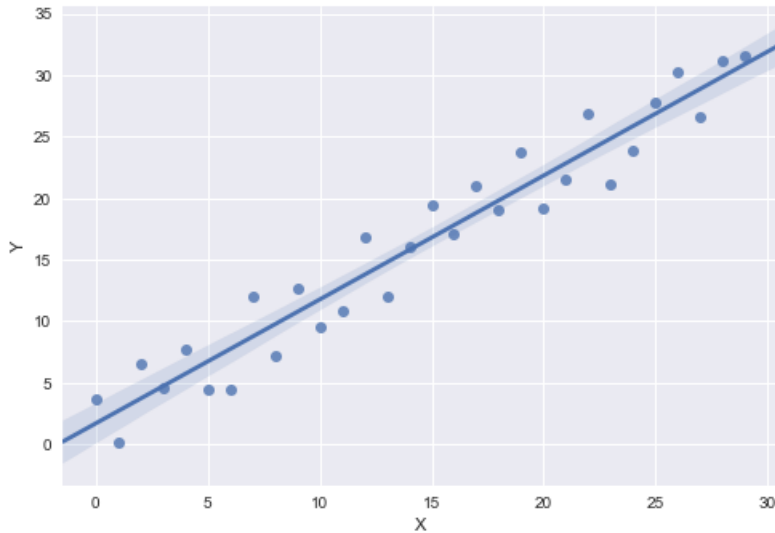
# Таблица для хранения стат.данных
from pandas import DataFrame

# поместим в таблицу как столбцы с заголовками X и Y
D = DataFrame( {'X':X, 'Y':Y} )

# Построим диаграмму рассеивания
seaborn.regplot(x='X', y='Y', data=D);

plt.show()
```


Диаграмма рассеивания



Outline

Введение

DataFrame

Линейная алгебра

Полиномы

Дифференциальные уравнения

Численные методы

Минимизация функции

Метод наименьших квадратов

Интерполяция

Интегрирование

Дифференцирование

Теория вероятностей и статистика

Гистограмма

Корреляция и диаграмма рассеивания

Диаграмма размаха

Символьные вычисления

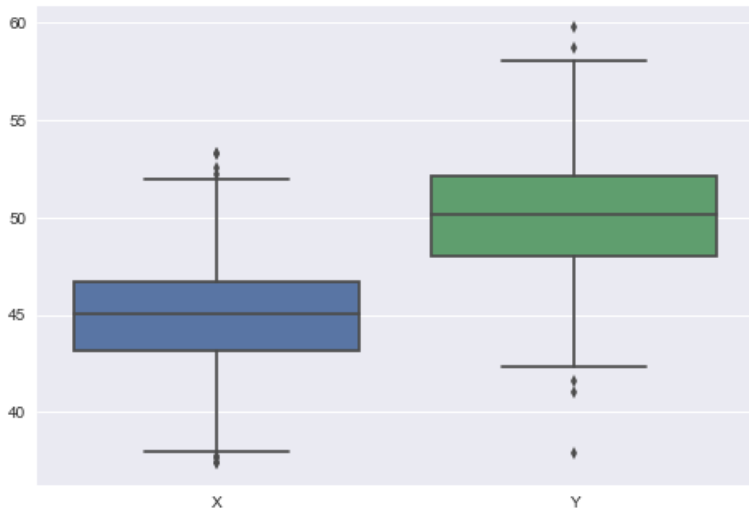
О быстройдействии

Ссылки и литература

Диаграмма размаха ("Ящик с усами")

```
X = [ sum([random() for i in range(90)])  
      for j in range(1000) ]  
Y = [ sum([random() for i in range(100)])  
      for j in range(1000) ]  
  
D = DataFrame( {'X' : X, 'Y':Y} )  
seaborn.boxplot(D)  
plt.show()
```

Диаграмма размаха ("Ящик с усами")



О визуализации статистический данных и проверке гипотез с помощью Python:

nahlogin.blogspot.ru/2016/01/pandas.html

Outline

Введение

DataFrame

Линейная алгебра

Полиномы

Дифференциальные уравнения

Численные методы

- Минимизация функции

- Метод наименьших квадратов

- Интерполяция

- Интегрирование

- Дифференцирование

Теория вероятностей и статистика

- Гистограмма

- Корреляция и диаграмма рассеивания

- Диаграмма размаха

Символьные вычисления

О быстройдействии

Ссылки и литература

Символьные вычисления

Символьные вычисления — это преобразования и работа с математическими равенствами и формулами как с последовательностью символов.

Системы символьных вычислений (их так же называют системами компьютерной алгебры) могут быть использованы для символьного интегрирования и дифференцирования, подстановки одних выражений в другие, упрощения формул и т. д.

SymPy

```
from sympy import *  
# символ  
x = symbols('x')  
a, b, c = symbols('a,b,c')  
  
# для красивого отображения математических  
# выражений в Jupyter  
init_printing(use_unicode=False,  
              wrap_line=False, no_global=True)
```

asmeurer.com/sympy_doc/dev-py3k/tutorial/tutorial.ru.html algebraic-equations - Краткое руководство по sympy

SymPy

упрощение выражений

`(a+a+a+a).expand()` # $\rightarrow 4*a$

подстановка выражений

подстановка $x=a$

`(2*x+y-a).subs(x,a)` # $\rightarrow a + y$

вычисление с подстановкой значений

`(a+a+a+a).subs(a,3)` # $\rightarrow 12$

Решение уравнений

```
# решение уравнения относительно x  
solve(3*x + x - 5, x)
```

```
[ $\frac{5}{4}$ ]
```

Ответ представляется списком из объектов типа Rational.

Для преобразования в вещественное число используется метод `n()`

```
# решение уравнения относительно x  
roots = solve(3*x + x - 5, x)  
  
x1 = roots[0].n() # 1.25
```

Дифференцирование

```
# дифференцирование
```

```
diff( sin(2*x), x)
```

$2 \cos(2x)$

```
# вторая производная
```

```
diff( sin(2*x), x, 2)
```

$-4 \sin(2x)$

Outline

Введение

DataFrame

Линейная алгебра

Полиномы

Дифференциальные уравнения

Численные методы

- Минимизация функции

- Метод наименьших квадратов

- Интерполяция

- Интегрирование

- Дифференцирование

Теория вероятностей и статистика

- Гистограмма

- Корреляция и диаграмма рассеивания

- Диаграмма размаха

Символьные вычисления

О быстройдействии

Ссылки и литература

О быстродействии

Интерпретатор Python работает медленно

Библиотеки Python могут работать быстро

Вычисления нужно оптимизировать перед кодированием

Профилирование в Jupyter

Измерение среднего времени выполнения выражения³

```
def foo():  
    s = 0  
    for i in range(10**7):  
        s += sin(i)  
    return s
```

```
%timeit foo()
```

2.51 s \pm 58.3 ms per loop (mean \pm std. dev. of 7 runs,
1 loop each)

³ время зависит от загрузки процессора другими задачами

Профилирование в Jupyter

Правильное использование numpy может быть эффективнее традиционных циклов

```
def bar():  
    return np.sum(np.sin(np.arange(10**7)))
```

```
%timeit bar()
```

723 ms \pm 5.33 ms per loop (mean \pm std. dev. of 7 runs,
1 loop each)

<https://docs.scipy.org/doc/numpy-1.15.1/reference/ufuncs.html>

Outline

Введение

DataFrame

Линейная алгебра

Полиномы

Дифференциальные уравнения

Численные методы

- Минимизация функции

- Метод наименьших квадратов

- Интерполяция

- Интегрирование

- Дифференцирование

Теория вероятностей и статистика

- Гистограмма

- Корреляция и диаграмма рассеивания

- Диаграмма размаха

Символьные вычисления

О быстройдействии

Ссылки и литература

Ссылки и литература

- ▶ Numerical methods in engineering with Python 3 / Jaan Kiusalaas.
- ▶ try.jupyter.org
- ▶ Как использовать Jupyter (ipython-notebook) на 100%
- ▶ ВШЭ: курс Дифференциальные уравнения (с некоторым количеством примеров кода на Python)

Ссылки и литература

Ссылка на слайды

github.com/VetrovSV/Programming